AMATH 582 Homework 3

William Livernois (willll@uw.edu)

February 10, 2020

Abstract

Principal component analysis was combined with image processing techniques to analyze the motion of a mass (paint can) on a spring using data from 3 different camera angles. Image stabilization was conducted using image convolution to smooth transitions between frames and thresholds were used to isolate and track the paint can. Singular value decomposition (SVD) methods were used to generate a transformation matrix and determine the weighting of each component.

1 Introduction and Overview

Tracking was conducted on image data from 3 different cameras under 4 different experimental conditions. Background feature tracking was attempted by convolution of a sub-image onto each frame to create a reference point for horizontal (x) and vertical (y) displacement measurements. This reference was used to stabilize the image between frames, applying some smoothing to improve the tracking and reduce overall noise. To track the paint can between frames, a simple threshold on the gray-scale image was applied to each cropped frame. The center of mass was recorded and stored to the vector.

Using this camera setup, experiments were conducted tracking harmonic motion of a paint can on a spring. Four experiments were conducted as follows:

1. Harmonic motion on the axis of the spring
2. Experiment 1 but with added noise (camera shake)
3. Harmonic motion in two directions: axis of the spring and perpendicular pendulum motion
4. Experiment 3 but with an added rotational component

For each experiment, six vectors were collected (x and y data for each of the 3 cameras) and SVD was used to diagonalize the co-variance matrix. This transformation gave the principal components of the paint can motion, which was compared to the expected motion based on the known experimental setup.

2 Theoretical Background

Image convolution is frequently used for image processing (such as edge detection or blurring) by multiplying a kernel (matrix) at each point of the image, with a convolution C defined as

$$C_{mn} = \omega * I_{mn} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \omega_{i,j} I(m+i, n+j) \tag{1}$$

where I is an $N \times M$ image with $m \in [1, M]$, $n \in [1, N]$ and ω is an $(2a+1) \times (2b+1)$ kernel. Applying this directly to an image is a time consuming process ($(2a+1) \times (2b+1) \times M \times N$ operations) that can be sped up using a Fourier transform. This can be done by applying the convolution theorem which states that Equation 1 can be transformed into

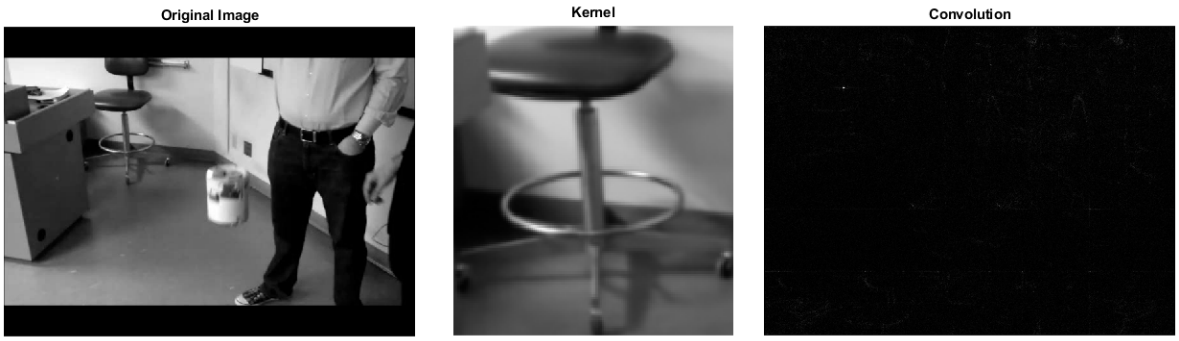

Figure 1: Image convolution to track location of the chair sub-image, which shows up a small white dot.

$$C = \omega * I = \mathcal{F}^{-1}[\mathcal{F}[\omega] \bullet \mathcal{F}[I]] \tag{2}$$

where the $\bullet$ represents a element-wise product of the matrices and $\mathcal{F}$ is the (discrete) Fourier Transform. If the kernel ω is set to a sub-image of the main image I the resulting convolution will show the density of that feature in the image, with the highest matrix element representing the location of that feature (an example is shown in Figure 1). Initially, SVD was used to create a kernel that represents the eigen-image of a cropped sub-image taken from several frames (see `eigenBucket.m` in Appendix B) but in the end an average of the images in each video file were used. The resulting convolution was used to determine any shifting of the image required for stabilization, and the transformation was applied.

After the paint can location was extracted using image the image threshold, the resulting vectors were stored as rows in a matrix X. These values were centered at zero (mean subtracted) to allow for rotation about the origin. Using SVD, the experimental matrix can be separated into the product $X = U\Sigma V^T$, with rotation matrices U and V^T, and diagonal singular value matrix Σ. This transformation allows us to determine the principal component matrix $Y = U^T X$ that has a co-variance matrix given by

$$\begin{aligned} Cov[Y] &= \frac{1}{n-1} YY^T = \frac{1}{n-1}(U^T X)(U^T X)^T = \frac{1}{n-1}(U^T U\Sigma V^T)(U^T U\Sigma V^T)^T \\ &= \frac{1}{n-1}(\Sigma V^T)(\Sigma V^T)^T = \frac{1}{n-1}(\Sigma V^T V\Sigma^T) = \frac{\Sigma^2}{n-1} \end{aligned} \tag{3}$$

which must be diagonalized because Σ is a diagonal matrix[2]. This matrix will be used to determine the weight of each principal component given by rows of Y (specified by Λ_i). In the case of these experiments, the principal components corresponded to the major (orthogonal) directions of motion.

3 Algorithm Implementation and Development

While the SVD calculations used to determine principal components were at the crux of this project, the main coding effort came from extraction of accurate, readable data from the camera footage. The algorithm implemented applies a threshold to each grayscale frame to leave behind objects high in brightness. The threshold was chosen so that the flashlight attached to the paint can was the only extracted object, and the centroid of the object was found using a cross product (weighted sum of the coordinates of each pixel). The centroid coordinates were stored as x and y data for each camera. The full algorithm is outlined below in Algorithm 1.

This simpler algorithm replaced the sub-image based convolution algorithm initially implemented, adding speed and reliability with the cost of some tracking noise that was later minimized by image stabilization. It was discovered in this first implementation that the paint can sub-image changes significantly within the

Algorithm 1: Paint Can Tracking Algorithm

```
Create empty X matrix with 6 rows for tracking data
for Cam in [Cam 1, Cam 2, Cam 3] do
  img = average of frames in video
  cropX, cropY = bounds to paint can tracking
  for frame in video do
    Convolve frame with kernel img
    Determine x-shift and y-shift for image stabilization
    if shifting necessary then
      Shift frame
    end if
    Crop frame by cropX and cropY
    Apply threshold and get centroid (x,y)
    Store (x,y) data to matrix X
  end for
  Center x and y values around the average
end for
return X matrix - x,y data for 3 cams
```

same video, due to motion blur and rotation, leading to an inaccurate result from sub-image convolution for some of the frames. Using the averaged image as a kernel for shifting still removed some of the noise from camera shake, and the only tracking parameter necessary for each experiment was the motion boundary of the paint can specified by the `cropX` and `cropY` variables.

principal components were extracted using `svd()` MATLAB function that uses a proprietary algorithm that is based on the `DGESVD` function in the LAPACK library[1]. To improve the accuracy of the principal component analysis, videos were synced by cropping any extra initial frames. Without this initial cropping, the phase shift of the harmonic motion between cameras lead to periodic artifacts in the principal components, and cropping indices were manually inserted to match up the videos.

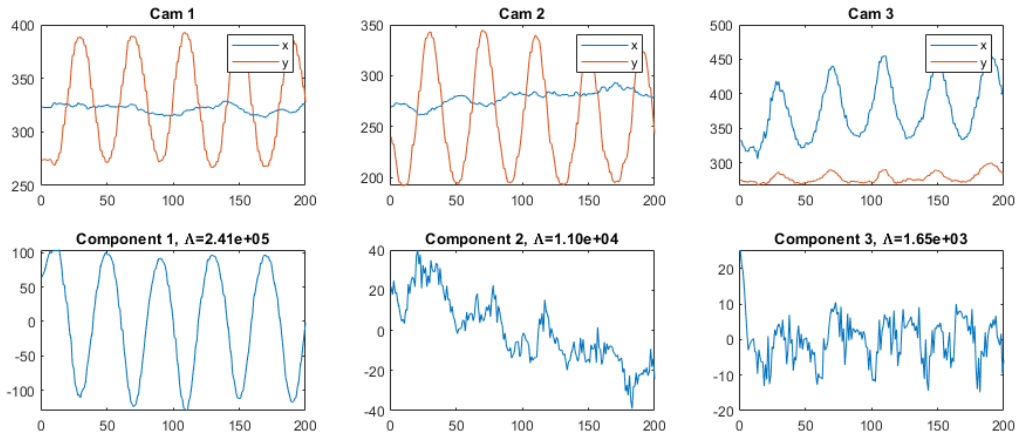

Figure 2: Paint can tracking x and y data (above) and first three principal components (below) with respective weighting values for Experiment 1

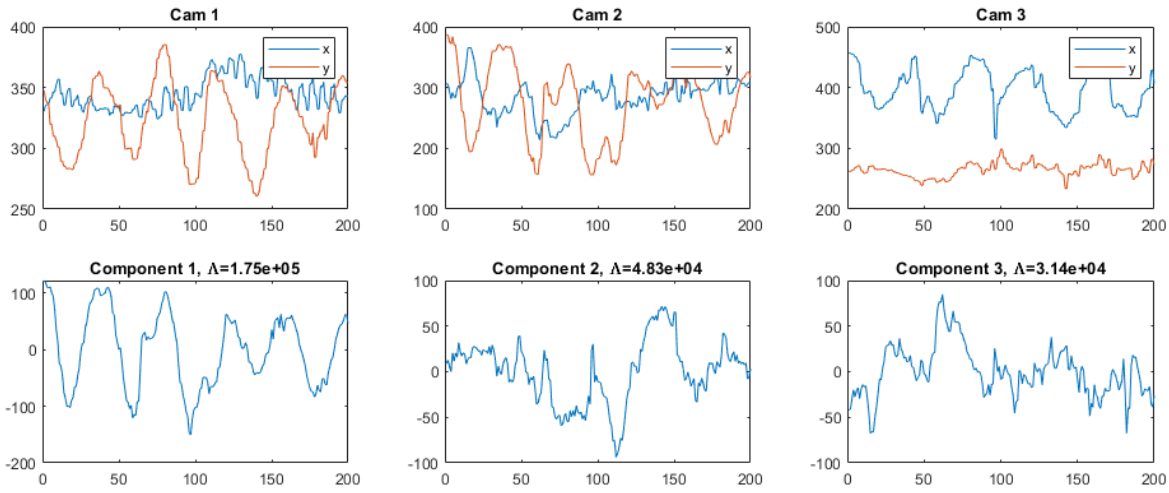

Figure 3: Paint can tracking x and y data (above) and first three principal components (below) with respective weighting values for Experiment 2

4 Computational Results

For the first experiment, the tracking data and first three components are shown in Figure 2. In this experiment, the cam 2 data was out of sync and so the first 10 frames were cropped. It is clear that the first principal component is the most relevant, representing the motion in one dimension, with a Λ value that is at least an order of magnitude higher than the other components. All other components appear as noise, which is especially depicted in the graph for Component 3. Looking at the first column of the U rotation matrix, we can see how the original X matrix was transformed to create the first component, with values of $[-0.0028, -0.5566, 0.0184, -0.6462, -0.5161, -0.0769]$ that correspond to higher relative weights of Cam 1 and 2 y-position and the Cam 3 x-position.

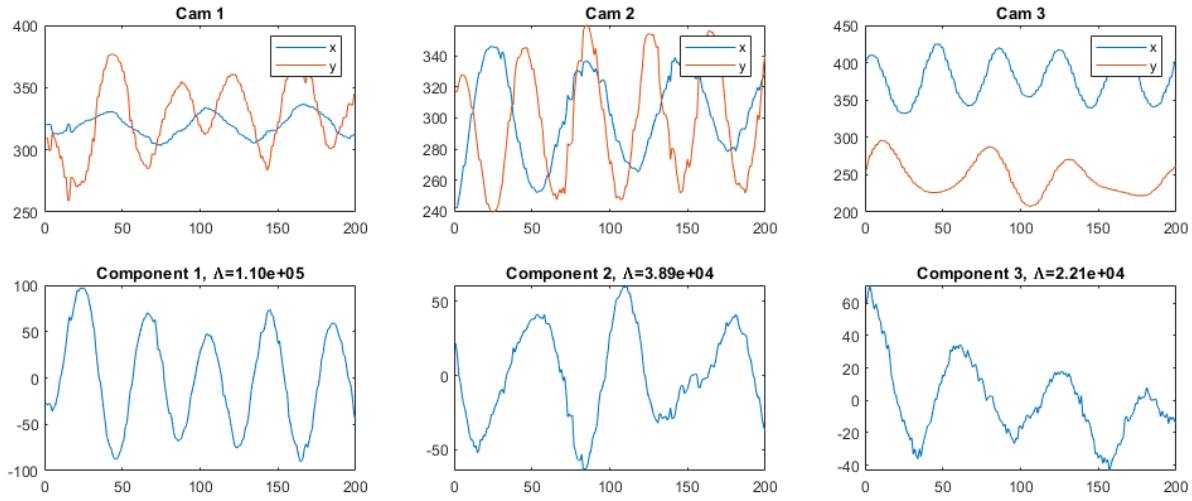

Figure 4: Paint can tracking x and y data (above) and first three principal components (below) with respective weighting values for Experiment 3

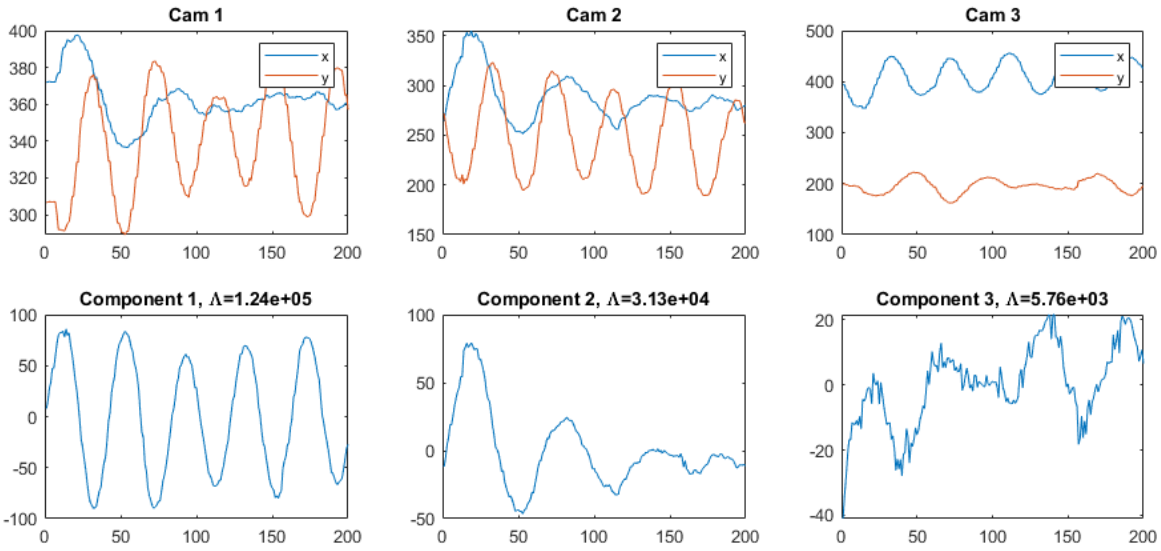

Figure 5: Paint can tracking x and y data (above) and first three principal components (below) with respective weighting values for Experiment 4

For the second experiment, the added noise from random motion of each camera is quite apparent in the tracking data shown in Figure 3. The first 17 frames of Cam 1 and 3 were cropped to sync up camera times. In this case, the noise of the applied data has caused the first component to only roughly match the harmonic motion, with noise spread equally across each principal component. Like in experiment 1, the values of the first column of the U rotation matrix showed a higher weight of the Cam 1 y-position, Cam 2 y-position, and Cam 3 x-position but with a less relative intensity (for instance, Cam 2 x-position had a weight of -0.1658).

The results from experiment 3 are shown in Figure 4 (note that image stabilization was turned off for these results). As in the previous two experiments, the first principal component clearly shows the harmonic motion in the direction of the spring using higher weights from the Cam 1 y-position, Cam 2 y-position, and Cam 3 x-position. However, the secondary motion direction is split between the following two components, with both components showing higher weights of the Cam 3 y-position (0.4806 and -0.3619 for component 2 and 3 respectively) and similar assigned lambda values (only off by a factor of 2.5).

In the last experiment, experiment 4, there were 3 degrees of motion, and the resulting tracking data and principal components are shown in Figure 5 (image stabilization was also turned off for these results). In this case, syncing of the second camera (cropping out the first 7 frames) was especially important to allow fully recovery of the 3 orthogonal motion directions shown by the first three principal components. It is clear that the spring axis motion and pendulum motion are captured by component 1 and 2 respectively. The rotation of the bucket is likely captured by the third principal component but the noise of tracking (and geometry of the motion) makes it harder to distinguish as a periodic signal.

5 Summary and Conclusions

The resulting principal components determined from each experiment almost exactly matched up with the expected movement from the experimental setup. Syncing of the videos was shown in all examples to be important, because any phase shift appearing between camera data lead to a new orthogonal direction of motion that was extracted as a principal component. Random noise added independently to each camera, as was done in experiment 2, did have an impact on the resulting principal components because the process

of diagonalizing the covariance matrix did not remove noise. If one were to visualize the dataset in two dimensions as an ellipse of points with vertices in the direction of the first principal component, adding Gaussian noise would make the resulting data set circular, drowning out the principal components.

Image stabilization using convolution had a limited impact on the precision of the data, and was only a useful addition to the first two experiments. Other more robust tracking algorithms use general metrics such as corner/edge detection or L-moments between images to track features between frames. The MATLAB image processing toolbox has several of these functions, including the Kanade-Lucas-Tomasi feature tracker. This method uses SVD to trace the path of points between image frames[3] and can be easily implemented with the camera data not only to track the paint can but also to create a reference point for image stabilization.

From our results it is possible to extract the underlying physics of the experiments. Though the equations of motion were not explicitly derived, we demonstrated that all motion was sinusoidal of the form

$$x(t) = A\exp(C_1 t)\cos(\omega t + C_2) \tag{4}$$

which is a solution to the differential equation $m\ddot{x} = kx - b\dot{x}$, as derived from Newton's law. In the case of the spring motion, the damping coefficient is very small ($b \approx 0$ and therefore $C_1 = 0$), whereas the pendulum motion has a more significant damping coefficient as shown by the exponential decay of the second principal component in Figure 5.

References

[1] Edward Anderson et al. *LAPACK users' guide*. SIAM, 1999.

[2] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[3] Bruce D Lucas, Takeo Kanade, et al. "An iterative image registration technique with an application to stereo vision". In: Vancouver, British Columbia. 1981.

Appendix A MATLAB Functions

The MATLAB functions used for these calculations are listed below:

- `[u, s, v] = svd(X)` returns the rotation matrices `u` and `v` and the diagonal matrix `s` as the singular value decomposition of `X` such that `X=u*s*v`.
- `m = diag(M)` returns the diagonal vector `m` of matrix `M`
- `[V,D] = eigs(A, n, 'lm');` returns the first `n` eigenvalues as diagonal matrix `D` and first `n` eigenvectors as columns of `V`.
- `f = fft2(I)` returns the 2-D fast Fourier transform `F` of matrix `I`
- `im = im2gray(I)` converts RGB image `I` to greyscale image `im` with 8-bit unsigned integer element values (0-255).
- `im = imtranslate(I, [dx, dy])` shifts image `I` by `dx` columns and `dy` rows, replacing uncovered locations with the value 0.

Functions used for basic arithmetic or plot formatting are not included above, but the important figures are included in the Computation Results section above.

Appendix B MATLAB Code

All code is located in the "Homework 3" folder of the github repository (`https://github.com/wliverno/AMATH582`).

Several MATLAB files were used for this project, the main one being `HW3.m` which runs all four experiments:

```
close all

%Experiment 1 - One direction of motion
load cam1_1.mat
load cam2_1.mat
load cam3_1.mat
vidFramesList={vidFrames1_1, vidFrames2_1(:,:,:,10:end), vidFrames3_1};
xCropList ={250:400, 200:380, 150:600};
yCropList ={1:480, 180:450, 225:350};
[lambda1, Y1, U1] = camAnalysis1(vidFramesList,xCropList,yCropList);

%Experiment 2 - One direction of motion with noise
load cam1_2.mat
load cam2_2.mat
load cam3_2.mat
xCropList ={250:400, 200:400, 175:600};
yCropList ={1:480, 140:420, 200:350};
vidFramesList={vidFrames1_2(:,:,:,17:end), vidFrames2_2, vidFrames3_2(:,:,:,17:end)};
figure;
[lambda2, Y2,  U2] = camAnalysis1(vidFramesList,xCropList,yCropList);

%Experiment 3 - Two directions of motion
load cam1_3.mat
load cam2_3.mat
load cam3_3.mat
vidFramesList={vidFrames1_3(:,:,:,16:end), vidFrames2_3, vidFrames3_3(:,:,:,6:end)};
figure;
xCropList ={250:400, 225:400, 170:600};
yCropList ={1:480, 150:450, 200:400};
[lambda3, Y3, U3] = camAnalysis1(vidFramesList,xCropList,yCropList);

%Experiment 4 - Three directions of motion
load cam1_4.mat
load cam2_4.mat
load cam3_4.mat
vidFramesList={vidFrames1_4, vidFrames2_4(:,:,:,7:end), vidFrames3_4};
figure;
xCropList ={250:400, 225:400, 200:600};
yCropList ={1:480, 150:450, 100:300};
[lambda4, Y4, U4] = camAnalysis1(vidFramesList,xCropList,yCropList);
```

The file `camAnalysis1.m` is a function that does all of the paint can tracking and principle components analysis given the 3 input video files and cropping information:

```
function [lambda, Y, U] = camAnalysis1(vidFramesList, xCropList, yCropList)
```

```
n=200;

%Loop through up to frame n and store values to matrix
X = zeros(6, n);
figure;
for j = 1:3
    vidFrames = cell2mat(vidFramesList(j));
    imgf = fft2(double(im2gray(vidFrames(:,:,:,1))));
    for i=2:n
        imgf=imgf+fft2(double(im2gray(vidFrames(:,:,:,i))));
    end
    imgf=imgf/n;
    img = ifft2(imgf);
    [ly,lx] = size(img);
    xCrop=cell2mat(xCropList(j));
    yCrop=cell2mat(yCropList(j));
    x=zeros(1,n);
    y=zeros(1,n);
    for i=1:n
        %Get current frame
        frame = double(im2gray(vidFrames(:,:,:,i)));
        frame = frame*255/max(frame(:));

        %Image stabilization
        [refX, refY] = getLocRef(frame,img, 0);
        refX = refX-320;
        refY = refY-240;
        try
            frame=imtranslate(frame, [-refX -refY]);
        end
        %Get coordinates of paint can
        [x(i), y(i)] = getLocCan(frame, xCrop, yCrop, 240);

        %In case of tracking failure, use previous data point
        if isnan(x(i))&(i>1)
            x(i) = x(i-1);
        end
        if isnan(y(i))&(i>1)
            y(i) = y(i-1);
        end
        %subplot(2,3,j), imshow(uint8(frame)), drawnow; %View video
    end
    t=1:n;
    subplot(2,3,j), plot(t,x,t,y);
    legend('x','y');
    title(['Cam ', num2str(j)]);
    X(j*2-1, :) = x(1:n)-mean(x(1:n));
    X(j*2, :) = y(1:n)-mean(y(1:n));
end

%PCA Analysis
[U, S, V] = svd(X/sqrt(5));
lambda = diag(S).^2;
Y =U'*X;
```

```
    subplot(2,3,4), plot(t,Y(1,:)), title(['Component 1, \Lambda=', num2str(lambda(1), '%.2e')]);
    subplot(2,3,5), plot(t,Y(2,:)), title(['Component 2, \Lambda=', num2str(lambda(2), '%.2e')]);
    subplot(2,3,6), plot(t,Y(3,:)), title(['Component 3, \Lambda=', num2str(lambda(3), '%.2e')]);

end

%Track paint can within bounds using specified threshold
function [xloc, yloc] = getLocCan(img, xcrop, ycrop, thres)
    [XC, YC] = meshgrid(xcrop, ycrop);
    croppedImg = double(img(ycrop, xcrop));
    croppedImg(croppedImg<=thres) = 0;
    xloc=sum(croppedImg.*XC, 'all')/sum(croppedImg, 'all');
    yloc=sum(croppedImg.*YC, 'all')/sum(croppedImg, 'all');
end

%Track kernel (img2) within image (img), return expected location with
%thres cutoff
function [xloc, yloc] = getLocRef(img, img2, thres)
    w = conj(fft2(img2));
    f = fft2(double(img));
    cnv = f.*w;
    cnv = cnv./abs(cnv);
    cnv = abs(ifft2(cnv));
    cnv(:,1) = 0;
    cnv(:,end) = 0;
    cnv(1,:) = 0;
    cnv(end,:) = 0;
    locGraph = cnv/max(cnv(:));
    locGraph = locGraph.*(locGraph>thres);
    [ly,lx] = size(img);
    [X, Y] = meshgrid(1:lx, 1:ly);
    xloc = round(sum(locGraph.*X, 'all')/sum(locGraph, 'all'));
    yloc = round(sum(locGraph.*Y, 'all')/sum(locGraph, 'all'));
    if isnan(xloc)
        xloc = 0;
    end
    if isnan(yloc)
        yloc=0;
    end
end
```

The file `eigenBucket.m` is a helper method for feature tracking that generates an "eigen-image" from cropping features across multiple frames in a video file. This method implements SVD using the `eigs()` MATLAB function:

```
function EB = eigenBucket(movie, numImages, skip)
    % Get Data Set by cropping images
    maxSubX = 0;
    maxSubY = 0;
    [lx,ly,~,n] = size(movie);
    bucketList = zeros(lx, ly, numImages);
    for i=1:numImages
        imshow(im2gray(movie(:,:,:,i*skip))), drawnow;
```

```
        subim = imcrop;
        clf
        [subX, subY] = size(subim);
        bucketList(1:subX,1:subY, i) = double(subim);
        if subX>maxSubX
            maxSubX = subX;
        end
        if subY>maxSubY
            maxSubY = subY;
        end
    end
    bucketListCropped = bucketList(1:maxSubX, 1:maxSubY, :);

    %Singular Value Decomposition
    A = zeros(numImages, maxSubX*maxSubY);
    for i=1:numImages
        A(i, :) = reshape(bucketListCropped(:, :, i), 1, maxSubX*maxSubY);
    end
    [V,D] = eigs(A'*A, 20, 'lm');
    d = diag(D);
    EBVec = abs(V*d)/max(abs(V*d));

    %Write eigenbucket as image
    EB = zeros(lx, ly);
    EB(1:maxSubX, 1:maxSubY) = reshape(EBVec*255,maxSubX, maxSubY);
    EB = uint8(EB);
    imshow(EB);
end
```
